4 Using DSPower®-HWLib and DSPower-SPLib Software

Overview

[@j Hardware Library, Signal Processing Library]

This section provides information needed to effectively use DSPower-HWLib and DSPower-SPLib software, including:

- background information about the way HWLib and SPLib work
- using the Hardware Manager and Engine Manager DLLs,
- using HWLib and SPLib, including acquiring/generating analog I/O data and calling DSP/math functions
- creating programs which use DSPower-HWLib and DSPower-SPLib DLLs

For a concise description and explanation of individual function calls inside all DSPower-HWLib and DSPower-SPLib DLLs, see Signalogic DSP Software Reference Guide section 2.2, DLL Functions, to find the function of interest. Also, Function Alphabetic List and Cross-Reference, in the overview section of Signalogic DSP Software Reference Guide chapter 2, Function Reference, contains an alphabetized list of functions.

For more information about media and disk space requirements to use DSPower-HWLib and DSPower-SPLib software, see Signalogic DSP Software System Guide sections 3.1.2, Media and Disk Space Requirements, and 3.1.3, Documentation and Media You Should Have Received.

For information about installing DSPower-HWLib and DSPower-SPLib software, see Signalogic DSP Software System Guide sections 3.2.2, Installing DSPower Series Software, and 3.2.3, Configuration and Checkout of DSPower-HWLib and DSPower-SPLib Software.

Main Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software

Creating Programs for DSPower-HWLib and DSPower-SPLib

DSPower-SPLib Function Overview

Acquiring Data Using DSPower-HWLib Software

Using Engine Manager

Using Hardware Manager

4.1 Architecture of DSPower-HWLib and DSPower-SPLib Software

DSPower-HWLib software consists of three (3) DLLs and several support utilities. The DLLs are:

- enmgr.dll (Engine Manager)

- hwmgr.dll (Hardware Manager)
- hwlib.dll (Hardware Library)

DSPower-SPLib software also consists of support files and programs and three DLLs:

- enmgr.dll (Engine Manager)
- hwmgr.dll (Hardware Manager)
- splib.dll (Signal Processing Library)

The Engine Manager and Hardware Manager DLLs are common to both DSPower-HWLib and DSPower-SPLib software. These DLLs are also used by DSPower-Block Diagram software. Essentially, Engine Manager is responsible for controlling and managing DSP Engines transparently to the calling application, and Hardware Manager is responsible for managing the configuration of all DSP/analog hardware installed in the system. For more information, see chapter 7, Using Engine Manager, and chapter 8, Using Hardware Manager.

Related Topics

General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software

Creating Programs for DSPower-HWLib and DSPower-SPLib

DSPower-SPLib Function Overview

Acquiring Data Using DSPower-HWLib Software

Using Engine Manager

Using Hardware Manager

4.1.1 Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

As with file management and window management in Windows, the concept of handles are used throughout DSPower-HWLib and DSPower-SPLib software. Specifically, the DSEngineOpen and DSAssignBoard calls return unique handles, which implicitly represent the information and parameters used to make these calls. The handles can then be used to simplify the calling convention for many other library calls, and to allow the system to produce profile, status, and error data as needed.

Engine Handles

Engine handles are returned by the DSEngineOpen call. The returned engine handles can then be used in DSAssignBoard calls, which assign specific hardware types for use by the engine.

Board Handles

Board handles are really "DSP/analog hardware handles", but board handle is an easier term to use in the DSPower-HWLib API. Unique board handles are created by the DSAssignBoard call, and can then be used to access DSP/analog hardware. Examples of DSPower-HWLib calls which require board handles include DSInitBoard, DSResetBoard, DSRunBoard, DSPutMem and DSGetMem, etc.

"DSxxxBoard" calls also have "DSxxxProcessor" equivalents. These calls can used to refer to specific processors on a multiprocessor DSP/analog hardware type. Examples include DSLoadFileProcessor, DSResetProcessor, DSRunProcessor, etc. When used, these calls affect only the processor to which the handle is assigned. The first parameter of these calls, like their DSxxxBoard counterparts, is a board handle. The second parameter of these calls is a "processor number", which is actually a value treated as a series of bits, where each bit refers to a processor (bit 0 == processor 0, bit 1 == processor 1, etc.). This method allows any valid combination of processors to be specified in DSxxxProcessor calls.

Notes

1) DSxxxBoard calls to multiprocessor hardware types affect all processors simultaneously. For example, DSLoadFileBoard loads the same DSP Program File to all processors, and DSRunBoard starts all processors simultaneously (assuming the multiprocessor DSP/analog hardware in question supports this mode).

2) DSxxxProcessor calls will return an error if a processor is specified which is out of range of what is actually on the hardware.

3) The current limit on both engine and board handles is sixteen (16).

Related Topics

Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Callback Messages in DSPower-HWLib and DSPower-SPLib Software

Error and Exception Handling in DSPower-HWLib and DSPower-SPLib Software

Performance Considerations in DSPower-HWLib and DSPower-SPLib Software

General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software

Creating Programs for DSPower-HWLib and DSPower-SPLib

DSPower-SPLib Function Overview

Acquiring Data Using DSPower-HWLib Software

Using Engine Manager

Using Hardware Manager

4.1.2 Callback Messages in DSPower-HWLib and DSPower-SPLib Software

DSPower-HWLib software allows user-specified window or control handles to be "registered" to several types of DSP Engine callback

messages. This is done using the DSRegisterEngineMsgWnd function, which is located in Engine Manager (enmgr.dll). Once registered, window or control handles receive callback messages containing information about, and indicating the status of, ongoing engine and DSP/analog hardware activity.

Below are the current callback message types available in DSPower-HWLib and DSPower-SPLib software:

- engine status messages
- program status messages
- buffer ready messages.
- semaphore messages
- statistics and parameter update messages

A window or control handle must be registered for each of the above callback message types before they can be sent by DSP Engines. Engine and program status messages are handled by Engine Manager (enmgr.dll), and buffer ready, semaphore, and statistics and parameter update messages are handled by either DSPower-HWLib (hwlib.dll) or DSPower-SPLib (splib.dll).

For detailed information about callback message types, and using the DSRegisterEngineMsgWnd function to enable window or controls to receive these messages, see section 7.4, Registering Callback Messages from DSP Engines.

For source code examples, see the following in section 4.2.4, Example Code: DSPower-HWLib and DSPower-SPLib Software:

Example C/C++ Code: Creating a Callback Window

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Example C/C++ Code: Unregistering Callback Handles to Receive Update Messages

Example C/C++ Code: Processing Buffer Ready Messages

Also, the dscope.cpp example source code file included with DSPower-HWLib software implements a digital oscilloscope example which shows the use of DSRegisterEngineMsgWnd and processing subsequent buffer ready messages.

Related Topics

Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Callback Messages in DSPower-HWLib and DSPower-SPLib Software

Error and Exception Handling in DSPower-HWLib and DSPower-SPLib Software

Performance Considerations in DSPower-HWLib and DSPower-SPLib Software

General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software

Creating Programs for DSPower-HWLib and DSPower-SPLib

DSPower-SPLib Function Overview

Acquiring Data Using DSPower-HWLib Software

Using Engine Manager

Using Hardware Manager

4.1.3 Error and Exception Handling in DSPower-HWLib and DSPower-SPLib Software

The DSPower-HWLib and DSPower-SPLib error handling scheme is straightforward. Most library calls return a non-zero value on success, and a zero (NULL) on failure. When calls into hwlib.dll (or splib.dll) fail, the DSGetHWLibErrorStatus (or DSGetSPLibErrorStatus) and DSGetEngineErrorStatus calls can be used to attempt to learn more information about the most recent failure. These calls return error and status conditions associated with DSP/analog hardware and DSP/math functions.

Generally, at any given time both of these calls will return the same value. However, there is an important difference: the latter retrieves error conditions associated with a particular engine, while the former

returns an overall DLL value. Thus, if more than one engine is open at one time, it is possible that the DSGetHWLibErrorStatus (or DSGetSPLibErrorStatus) value could be overwritten as a result of consecutive unrelated engine calls.

For error and status conditions associated with engine problems, the DSGetEngMgrErrorStatus and DSGetEngineErrorStatus routines (located in enmgr.dll) should be used. For error and status conditions associated with the Hardware Setup File or Hardware Selector, the DSGetHWMgrErrorStatus routine (located in hwmgr.dll) should be used.

NOTE: The DSGetEngineErrorStatus call is actually inside Engine Manager (enmgr.dll). Note also that Engine Manager maintains an "error/status listbox" window which can be made visible using the DSShowEngMgrStatusWindow call. This status window shows a history of both status and error conditions, including both those resulting from DSPower series DLL calls and generated by DSP Engines. For complete information on error handling in Engine Manager, see section 7.5, Error Handling in Engine Manager.

Below is an example showing error investigation after a DSAssignBoard failure:

```
if (!DSAssignBoard(hEngine, szBoardDesignator, uBusType, uIOBaseAddr,
uMemBaseAddr)) {

    nErrorType = DSGetEngineErrorStatus(hEngine);
    if (nErrorType == NULL) nErrorType = DSGetEngMgrErrorStatus();
    if (nErrorType == NULL) nErrorType = DSGetHWLibErrorStatus();
    ...do error processing...
}
```

In the above example, the actual return value of DSAssignBoard would be NULL in the case of failure, since a non-NULL value would be a valid board (hardware) handle. Thus, to find out more information, the DSGetEngineErrorStatus, DSGetEngMgrErrorStatus, and DSGetHWLibErrorStatus calls are checked. Note that DSGetEngineErrorStatus requires a valid engine handle as an input parameter. Thus, for problems related to a DSEngineOpen call, the DSGetEngineErrorStatus function should be avoided.

Below is a partial list of error values which can be either returned by DSGetEngineErrorStatus or DSGetHWLibErrorStatus, along with a brief description of their meaning. These constants can be found mostly in the hwlib.h file and partly in the dllerr.h file. For a list of

Engine Manager and Hardware Manager error values, see sections 7.5.1, DLL Error Handling in Engine Manager, and 8.4.1, DLL Error Handling in Hardware Manager. For a complete list of all DSPower series software error conditions by error number, see Error and Status Condition Numeric List and Cross Reference, in Signalogic DSP Software Reference Guide chapter 7, Error and Status Conditions.

dllerr.h or hwlib.h constant	Summary of Meaning
DSDSPFILENOTFOUND	Typically occurs after DSLoadFileBoard or DSLoadFileProcessor call. Indicates a DSP Program File could not be found.
DSHARDWARENOTRESPONDING	Typically occurs after a DSInitBoard call. Indicates DSP/analog hardware not present or not installed correctly.
DSINVALIDBOARDHANDLE	Board handle was invalid.
DSCOULDNOTOPENTEMPLATEFILE	A template file needed by a high-level function could not be opened.
DSCOULDNOTCREATESOURCEFILE	A DSP Engine source code file being generated by a high-level function could not be created.
DSCREATINGENGINEPROGRAMERROR	An error occurred when compiling or linking a DSP Engine source code file(s).
DSUNABLETOALLOCATEMEMORY	Memory for the function could not be allocated
DSUNABLETOREADTEMPLATEFILE	An error occurred reading a template file needed b a high-level function.
DSERRORINDATAPARAM	A parameter passed to a high level function in a data structure was invalid.
DSBOARDDOESNOTSUPPORTCALL	The hardware type does not

	support the call.
DSINVALIDWORDLENGTH	A wordlength parameter passed to a memory transfer function was invalid.
DSINVALIDMEMORYTYPE	A memory type parameter passed to a memory transfer function was invalid.
DSINVALIDBOARDDESIGNATOR	The board designator string was not recognized, or not listed in the Hardware Setup File.
DSBOARDNOTINHWSETUPFILE	The board handle or designator is no longer listed in the Hardware Setup File
DSALLBOARDHANDLESALLOCATED	No more board handles can be allocated.
DSINVALIDPROCESSORNUMBER	The processor number was out of range vs. the number of processors on the hardware.
DSCOULDNOTINITIALIZEDLL	Hwmgr.dll itself could not be initialized.

Note that in C/C++ programs, it is safest to treat the above error definitions as integer types, since some values are "positive" and some "negative". See Error and Status Condition Numeric List and Cross Reference, in Signalogic DSP Software Reference Guide chapter 7, Error and Status Conditions, for more information about this grouping.

Related Topics

Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Callback Messages in DSPower-HWLib and DSPower-SPLib Software

Error and Exception Handling in DSPower-HWLib and DSPower-SPLib Software

Performance Considerations in DSPower-HWLib and DSPower-SPLib Software

General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software

Creating Programs for DSPower-HWLib and DSPower-SPLib

DSPower-SPLib Function Overview

Acquiring Data Using DSPower-HWLib Software

Using Engine Manager

Using Hardware Manager

4.1.4 Performance Considerations in DSPower-HWLib and DSPower-SPLib Software

Like any software package, there are several items and tradeoffs to consider when attempting to optimize the performance of DSPower-HWLib and DSPower-SPLib software. Below are some areas to keep in mind.

Performance Considerations Topics

Continuous Disk Transfer Using DSPower-HWLib

Changing DSP Programs Using DSPower-HWLib

Related Topics

Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Callback Messages in DSPower-HWLib and DSPower-SPLib Software

Error and Exception Handling in DSPower-HWLib and DSPower-SPLib Software

Performance Considerations in DSPower-HWLib and DSPower-SPLib Software

General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software

Creating Programs for DSPower-HWLib and DSPower-SPLib

DSPower-SPLib Function Overview

Acquiring Data Using DSPower-HWLib Software

Using Engine Manager

Using Hardware Manager

Continuous Disk Transfer Using DSPower-HWLib

When implementing continuous transfer to display or disk using DSPower-HWLib software, the following are points to consider:

1) Video and display considerations mentioned in Signalogic DSP Software System Guide section 1.4.3, Video and Display Considerations, are applicable. In addition, because Windows 3.x graphics is not known for its speed, the number of points and window size makes a big difference. For example, plotting a full-scale waveform on a maximized window will take much longer to draw than if the waveform amplitude is small or the window size is reduced. Furthermore, if there are more x-axis points than the window is wide, you may want to consider an algorithm that reduces the actual number of x-axis points by averaging or taking the maximum amplitude of each group of points that map to the same display pixel.

2) When transferring data continuously between disk and DSP/analog hardware, bigger is better for buffer sizes. Especially with caching utilities such as SMARTDRV.EXE, or with 32-bit disk access enabled, there are many possibilities for disk drive delays. Sometimes it is not at first obvious why these "disk drive performance enhancers" can have detrimental effects on long-period continuous data transfer; for some interesting observations, see DSP/Analog Hardware Performance Considerations, in Signalogic DSP Software System Guide section 1.4.2, Performance Considerations. In general, it is true that bigger

buffers help absorb disk drive delays. Note that when using a Hypersignal-Macro series software as a DSP Engine, the maximum buffer size possible is currently 32768.

3) DSP/analog hardware data transfer speed is critical. There are a wide variety of methods of data transfer implemented on hardware types that are in common use, with an equally wide variety of transfer speeds. Essentially, DSP/analog hardware which offers some sort of inherent "host port" method of data transfer provide the fastest rates; additionally, these methods usually operate reliably regardless of whether the DSP device is running. Hardware with fully dual-ported SRAM is also a fast option, but can entail some unpleasant system problems if a memory-mapped addressing method is used. For a review of DSP/analog hardware data-transfer methods, see chapter 2, Hardware Considerations, in the Hypersignal-Macro Series Hardware Reference Guide.

Performance Considerations Topics

Continuous Disk Transfer Using DSPower-HWLib

Changing DSP Programs Using DSPower-HWLib

Related Topics

Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Callback Messages in DSPower-HWLib and DSPower-SPLib Software

Error and Exception Handling in DSPower-HWLib and DSPower-SPLib Software

Performance Considerations in DSPower-HWLib and DSPower-SPLib Software

General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software

Creating Programs for DSPower-HWLib and DSPower-SPLib

DSPower-SPLib Function Overview

Acquiring Data Using DSPower-HWLib Software

Using Engine Manager

Using Hardware Manager

Changing DSP Programs Using DSPower-HWLib

The fastest way to start a new DSP function is to already have it loaded in DSP board memory, and simply do the following:

1. change the reset vector value in the DSP code, using DSPutMem
2. reset the DSP/analog hardware, using DSResetBoard or DSResetProcessor

Another alternative is to download new DSP code to an area of DSP program memory currently not being used, and again change the reset vector and reset the board. Note that this method is only possible if the DSP/analog hardware supports program memory access from the host PC while the DSP is running. In the worst case, the DSP must be stopped, new code downloaded, and the DSP restarted.

Performance Considerations Topics

Continuous Disk Transfer Using DSPower-HWLib

Changing DSP Programs Using DSPower-HWLib

Related Topics

Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software

Callback Messages in DSPower-HWLib and DSPower-SPLib Software

Error and Exception Handling in DSPower-HWLib and DSPower-SPLib Software

Performance Considerations in DSPower-HWLib and DSPower-SPLib Software

General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software
Creating Programs for DSPower-HWLib and DSPower-SPLib
DSPower-SPLib Function Overview
Acquiring Data Using DSPower-HWLib Software
Using Engine Manager
Using Hardware Manager

4.2 Creating Programs for DSPower-HWLib and DSPower-SPLib

The best advice that can be given for creating your first programs which use DSPower-HWLib or DSPower-SPLib software is to look at the source code and project file examples provided with the software. That being said, this section and its related topics discuss the creation of programs using DSPower-HWLib and DSPower-SPLib software in detail.

Project files are included with DSPower-HWLib and DSPower-SPLib software for Borland C/C++ v3.1 and v4.5, Microsoft Visual C/C++ v1.5, and WATCOM C/C++[32] v10.0. These project files and compiler versions have been tested at Signalogic. You should be able to load these project files directly into your IDE, and compile them immediately. From there, it should be straightforward to either create modified versions of these projects, or create new projects that incorporate the necessary elements.

Note that the demo programs all invoke the Hardware Selector, which allows user-selection of the DSP/analog hardware type on which to run the demos. The Hardware Selector is provided mainly for convenience, and does not necessarily need be used in user-defined applications. The return value from the DSShowHardwareSelectorDlg call is a Board Designator string, which can then be used in DSAssignBoard calls to generate a board handle. Alternatively, DSAssignBoard calls can easily be "hardcoded" for a specific hardware type.

The following basic elements might typically be found in a Windows application using DSPower-HWLib or DSPower-SPLib software. The steps marked with ‡ can be performed in either the WinMain section of the application, or in a WM_CREATE message of a window procedure:

1. ‡ optional: call DSShowHardwareSelectorDlg to obtain a user-specified Board Designator
2. ‡ open a DSP Engine using the DSEngineOpen call
3. ‡ assign a board handle with DSAssignBoard using the engine handle returned by DSEngineOpen and the Board Designator returned by DSShowHardwareSelectorDlg; alternatively, "hardcode" the Board Designator string directly, and...
    4. ‡ perform a DSInitBoard call to verify the presence of the selected DSP/analog hardware and ensure it is responding at the correct I/O and/or memory base address(es)
    5. ‡ perform a DSResetBoard call
    6. ‡ perform a DSLoadFileBoard call to download a desired DSP Program file
    7. create a window to receive "buffer ready" messages from the DSP Engine
    8. respond to buffer ready callback messages to the window procedure and process data coming from or going to the DSP/analog hardware

or...

1. use the engine handle from DSEngineOpen in high-level functions such as DSAcquireWvfrmFile or DSGenerateWvfrmFile

For source code examples showing actual implementation of some of the above items, see section 4.2.4 below, Example Code: DSPower-HWLib and DSPower-SPLib Software.

Related Topics

General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software
Creating Programs for DSPower-HWLib and DSPower-SPLib
DSPower-SPLib Function Overview
Acquiring Data Using DSPower-HWLib Software
Using Engine Manager
Using Hardware Manager

4.2.1 Hardware Library and Signal Processing Library Access

[@k implicit loading, explicit loading]

To access Hardware Library and/or Signal Processing Library, the hwlib.dll and/or splib.dll libraries must be visible to your application's calling program. The easiest way to do this is to include the hwlib.lib and/or splib.lib files in your project. For C/C++ applications, please see the example project files, including Borland C/C++ v3.1 and v4.5, Microsoft Visual C/C++ v1.5, and WATCOM C/C++[32] v10.0, provided with DSPower-HWLib software. For MATLAB projects, please see the example .m files. Once included in your project file, and thus linked into your final executable, the .lib files cause their corresponding DLLs to be loaded and initialized when your executable first runs (actually, *before* any code in the executable runs). This is known as implicit loading.

Another way of accessing Hardware Library or Signal Processing Library is to load them from your application program dynamically, using the LoadLibrary call in the Windows API, followed by one or more GetProcAddress calls with specific function names as input parameters. This method is known as explicit loading, and is more complicated, requiring painstaking typecasting to create function prototype *pointers* which are set as a result of GetProcAddress calls. "Behind the scenes", this is also the technique used by MATLAB to load DLLs needed by calling .m programs--except that MATLAB simplifies the process by allowing DLL calls to look much like other MATLAB function calls.

Also, for C/C++ programs, the hwlib.h and/or splib.h include files must be specified along with other include files that may be needed by the calling application. Note that both the hwlib.h and splib.h include files expects the standard Windows 3.1 windows.h include file to have been previously specified.

For more information about using library and include files, see sections 4.2.2 and 4.2.3 below, Include Files for DSPower-HWLib and DSPower-SPLib, and .LIB Files for DSPower-HWLib and DSPower-SPLib.

Related Topics

Hardware Library and Signal Processing Library Access

Include Files for DSPower-HWLib and DSPower-SPLib

.LIB Files for DSPower-HWLib and DSPower-SPLib

Example Code: DSPower-HWLib and DSPower-SPLib Software

General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software

Creating Programs for DSPower-HWLib and DSPower-SPLib

DSPower-SPLib Function Overview

Acquiring Data Using DSPower-HWLib Software

Using Engine Manager

Using Hardware Manager

4.2.2 Include Files for DSPower-HWLib and DSPower-SPLib

[@k include files]

For C/C++ programs using DSPower-HWLib software, the first two, and possibly the third, of the following include files must be specified:

- hwlib.h or splib.h (Hardware Library or Signal Processing Library)

- enmgr.h (Engine Manager)
- hwmgr.h (Hardware Manager)

In some cases, hwmgr.h may not be necessary. This would occur if the calling application made no provision for operators to change or store parameters regarding DSP/analog hardware configuration, and thus made no calls to the Hardware Setup File.

The above include files needed by the calling application must be specified along with other Windows or standard C./C++ include files that may be required. Note that all of the above include files expect the standard Windows 3.1 include file windows.h to have been previously specified.

NOTE: For C/C++ development, make sure the include file directory list in your IDE contains the DSPower-HWLib subdirectory. The example project files supplied with DSPower-HWLib software assume default include file subdirectories of "hwlib", "toollib", and "helplib" under the root-level installation subdirectory, although toollib and helplib are present only for demo purposes.

Related Topics

Hardware Library and Signal Processing Library Access

Include Files for DSPower-HWLib and DSPower-SPLib

.LIB Files for DSPower-HWLib and DSPower-SPLib

Example Code: DSPower-HWLib and DSPower-SPLib Software

General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software

Creating Programs for DSPower-HWLib and DSPower-SPLib

DSPower-SPLib Function Overview

Acquiring Data Using DSPower-HWLib Software

Using Engine Manager

Using Hardware Manager

4.2.3 .LIB Files for DSPower-HWLib and DSPower-SPLib

[@k library files]

Example project files are included with DSPower-HWLib and DSPower-SPLib software for Borland C/C++ v3.1 and v4.5, Microsoft C/C++ v1.5, and WATCOM C/C++[32] v10.0. These project files include references to the following library files:

- hwlib.lib or splib.lib (Hardware Library or Signal Processing Library)
- enmgr.lib (Engine Manager)
- hwmgr.lib (Hardware Manager)

Either the hwlib.lib or splib.lib library files, and the enmgr.lib and hwmgr.lib files, must be specified in the project files of calling applications in order to allow compile-time linking and cause run-time implicit loading of the enmgr, hwmgr, and hwlib and/or splib DLLs. Even if no calls are made to the Hardware Manager DLL by the application, the hwmgr.lib file should still be specified in the project file, since the hwlib, splib, or enmgr DLLs may contain such calls.

Implicit loading under Windows means that any DLLs used by an application are automatically loaded and initialized when the application runs (this occurs before any code in the application itself executes).

NOTE: Enmgr.lib, hwmgr.lib, hwlib.lib and/or splib.lib library files are absolutely required in your project file if creating a new application. References to DSPower-HWLib or DSPower-SPLib calls will produce link-time errors if these library files are not "visible" in the application's project file. The example project files supplied with DSPower-HWLib software assume default library file subdirectories of "hwlib", "toollib", and "helplib" under the root-level installation subdirectory, although toollib and helplib are present only for demo purposes.

Related Topics

Hardware Library and Signal Processing Library Access

Include Files for DSPower-HWLib and DSPower-SPLib

.LIB Files for DSPower-HWLib and DSPower-SPLib
Example Code: DSPower-HWLib and DSPower-SPLib Software

General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software
Creating Programs for DSPower-HWLib and DSPower-SPLib
DSPower-SPLib Function Overview
Acquiring Data Using DSPower-HWLib Software
Using Engine Manager
Using Hardware Manager

4.2.4 Example Code: DSPower-HWLib and DSPower-SPLib Software

Below are several source code examples illustrating fundamental concepts in DSPower-HWLib and DSPower-SPLib software. These are presented here mainly for convenience, especially so for the online Help version of documentation, which allows an opportunity to jump back and forth between the examples and individual function descriptions in Signalogic DSP Software Reference Guide section 2.2, DLL Functions. Also, Function Alphabetic List and Cross-Reference, in the overview section of Signalogic DSP Software Reference Guide chapter 2, Function Reference, contains an alphabetized list of functions. For the most complete source code examples, please refer directly to the .cpp files included with DSPower-HWLib and DSPower-SPLib software.

DSPower-HWLib and DSPower-SPLib C/C++ Source Code Examples

Related Topics

Hardware Library and Signal Processing Library Access
Include Files for DSPower-HWLib and DSPower-SPLib
.LIB Files for DSPower-HWLib and DSPower-SPLib
Example Code: DSPower-HWLib and DSPower-SPLib Software

General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software
Creating Programs for DSPower-HWLib and DSPower-SPLib
DSPower-SPLib Function Overview
Acquiring Data Using DSPower-HWLib Software
Using Engine Manager
Using Hardware Manager

Example C/C++ Code: Include Statements for DSPower-HWLib

Below is an excerpt from the dscope.cpp source code example file included with DSPower-HWLib software. It shows a typical include file section which allows compilation of applications using the DSPower-HWLib software. Remember that referring to the prototypes inside the DSPower-HWLib DLL include files shown below will cause the necessary .lib (library) files to be needed in the application's project file at link-time; for more information on DSPower-HWLib and DSPower-SPLib .lib files, see section 4.2.3 above, .LIB Files for DSPower-HWLib and DSPower-SPLib.

```
// typical application-specific include files
#include <windows.h>
#include <stdlib.h>
#include <math.h>

// DSPower-HWLib DLL include files
#include "enmgr.h"   // Engine Manager
#include "hwmgr.h"   // Hardware Manager
#include "hwlib.h"   // Hardware Library

// more Signalogic DLL include files, but used only for user-interface in demo
#include "toollib.h"
```

```
#include "helplib.h"
#include "minmax.h"  // local versions of min/max, if needed
```

DSPower-HWLib and DSPower-SPLib C/C++ Source Code Examples

Example C/C++ Code: Include Statements for DSPower-HWLib

Example C/C++ Code: Initializing DSP/Analog Hardware

Example C/C++ Code: Creating a Callback Window

Example C/C++ Code: Processing Buffer Ready Messages

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Example C/C++ Code: Unregistering Callback Handles to Receive Update Messages

Example C/C++ Code: Acquiring Waveform Files

Example C/C++ Code: Playing Back Waveform Files

Example C/C++ Code: Initializing DSP/Analog Hardware

Below is a source code example similar to the dscope.cpp file, showing some engine and DSP/analog hardware initialization typical of working with DSPower-HWLib.

```
char szBoardDesignator[50];
HENGINE hEngine;
HBOARD hBoard;
DWORD dwFsMode, dwMemSize;
WORD wBoardClass;
UINT uMemArch;
float FsActual;

// exit WinMain if nothing to do in demo
if (DSShowHardwareSelectorDlg(NULL, szBoardDesignator) == IDCANCEL) goto
cleanup;

// open engine
hEngine = DSEngineOpen(DS_EO_HSM, NULL);

// first try to open Hypersignal-Macro or Hypersignal-Macro EX as engine
```

```
if (!hEngine) {

    // if that doesn't work, try Hypersignal-Acoustic
    hEngine = DSEngineOpen(DS_EO_HSA, NULL);

    if (!hEngine) { // still an error?

        short int err = DSGetHWLibErrorStatus();
        itoa(err, tmpstr, 10);
        MessageBox(NULL, tmpstr, "Dscope Demo:  DSEngineOpen failed",
        MB_OK);
        goto cleanup;
    }
}

// assign board handle:  engine handle, bd designator, bus type, IO/mem base addrs
hBoard = DSAssignBoard(hEngine, szBoardDesignator, NULL, NULL, NULL);

// initialize the board; make sure it's installed and responded
if (!DSInitBoard(hBoard)) {

    itoa(DSGetEngineErrorStatus(hEngine), tmpstr, 10);
    MessageBox(NULL, tmpstr, "Dscope Demo:  DSInitBoard failed", MB_OK);
    goto cleanup;
}

// interrogate engine for board type values
wBoardClass = DSGetBoardClass(hBoard);

// get memory architecture
uMemArch = DSGetMemArch(hBoard);
if (uMemArch == NULL) {

    MessageBox(NULL," ","Dscope Demo: DSGetMemArch failed", MB_OK);
    goto cleanup;
}

// find samp rate ctrl. reg. value and actual rate (closest rate possible to desired)
// 22.05 kHz sampling rate, assume simultaneous sampling

dwFsMode = DSCalcSampFreq(hBoard, 22050.0, NULL, NULL, &FsActual);

// load DSP Program File (usually a COFF file produced by DSP chip linker)
if (!DSLoadFileProcessor(hBoard, NULL, 0x01)) {  // load default file (proc 0 only)

    MessageBox(NULL, " ", "Dscope Demo:  problem loading file", MB_OK);
    goto cleanup;
}

// get the memory size, (note that this currently has to be done after LoadFile)
```

```
dwMemSize = DSGetMemSize(hBoard, 0x01);

// reset the DSP board (processor 0 only)
DSResetProcessor(hBoard, 0x01);

// send down some important variables (needed if a default Hypersignal DSP file is
being used)
...
...
DSPutHVarMem(hBoard, DSP_BOARDCLASS, wBoardClass & 0x00ff);
DSPutHVarMem(hBoard, DSP_BOARDSUBCLASS, wBoardClass >> 8);
...
DSPutHVarMem(hBoard, DSP_FSMODE, dwFsMode);
...

...initialize more DSP code variables, flags, buffers, etc. if necessary..

// scope window is born; turn the board loose...
DSRunProcessor(hBoard, 0x01);  // processor 0 only
```

DSPower-HWLib and DSPower-SPLib C/C++ Source Code Examples

Example C/C++ Code: Include Statements for DSPower-HWLib

Example C/C++ Code: Initializing DSP/Analog Hardware

Example C/C++ Code: Creating a Callback Window

Example C/C++ Code: Processing Buffer Ready Messages

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Example C/C++ Code: Unregistering Callback Handles to Receive Update Messages

Example C/C++ Code: Acquiring Waveform Files

Example C/C++ Code: Playing Back Waveform Files

Example C/C++ Code: Creating a Callback Window

Below is another source code example taken from dscope.cpp, showing partial code for creating a window and designating it to receive buffer ready messages:

```
// create window that can be used for callback messages
hwnd = CreateWindow(wndclass.lpszClassName,
        "Example DSPower-HWLib Test Program",
        WS_OVERLAPPEDWINDOW | WS_VSCROLL | WS_HSCROLL |
        WS_CLIPCHILDREN | WS_VISIBLE,
        rectScopePos.left,
        rectScopePos.top,
        nWidth,
        nHeight,
        NULL,
        NULL, // no menu
        hInstance,
        NULL);

ShowWindow(hwnd, nCmdShow);
UpdateWindow(hwnd);

// scope window is born; turn the board loose...
DSRunProcessor(hBoard, 0x01);  // processor 0 only

// register window to receive buffer ready messages, tell engine to wait for input data

DSRegisterEngineMsgWnd(hEngine, DS_REMW_SETDSPDATARDYMSG, hwnd);
DSWaitForBuffer(hBoard, 0, NULL, DS_WFB_POLLED);

// no choice now but to follow Alice...
while (GetMessage(&msg, NULL, 0, 0)) {

    TranslateMessage (&msg);
    DispatchMessage (&msg);
};
```

DSPower-HWLib and DSPower-SPLib C/C++ Source Code Examples

Example C/C++ Code: Include Statements for DSPower-HWLib

Example C/C++ Code: Initializing DSP/Analog Hardware

Example C/C++ Code: Creating a Callback Window

Example C/C++ Code: Processing Buffer Ready Messages

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Example C/C++ Code: Unregistering Callback Handles to Receive Update Messages

Example C/C++ Code: Acquiring Waveform Files
Example C/C++ Code: Playing Back Waveform Files

Example C/C++ Code: Processing Buffer Ready Messages

Below is a source code example from dscope.cpp showing some typical processing upon receiving a buffer ready message:

```
WORD wBuflen;
DWORD dwBufferBaseAddr;
int nCurBuf;
UINT uStatus;
int far* pBuf;

case WM_DSPENGINE_BUFRDY:  // buffer ready message sent by DSP engine

    pBuf = (int far*)GlobalLock(hBuf);  // lock memory to be used

    // transfer below illustrates two common types of DSP memory architectures

    if (DSGetMemArch(hBoard) == DS_GMA_VECTOR) {  // vector data memory

        if (nCurBuf == 0)  uStatus = DSGetMem(hBoard,
        DS_GM_VECTOR_DATA_X, dwBufferBaseAddr, DS_GM_SIZE16, pBuf,
        wBuflen);

        else  uStatus = DSGetMem(hBoard, DS_GM_VECTOR_DATA_Y,
        dwBufferBaseAddr, DS_GM_SIZE16, pBuf, wBuflen);
    }
    else { // linear data memory

        uStatus = DSGetMem(hBoard, DS_GM_LINEAR_DATA_RT,
        dwBufferBaseAddr+nCurBuf*wBuflen, DS_GM_SIZE16, pBuf, wBuflen);
    }

    GlobalUnlock(hBuf);  // let memory float

    if (!uStatus) MessageBox(NULL, "DSGetMem:  problem with point transfer",
    "Dscope Test Prog", MB_OK);

    nCurBuf ^= 1;  // switch buffers
    DSPutHVarMem(hBoard, DSP_HOSTBUFNUM, nCurBuf);// write new buffer #

    // may we have another? please...
    DSWaitForBuffer(hBoard, nCurBuf, NULL, DS_WFB_POLLED);
    break;
```

DSPower-HWLib and DSPower-SPLib C/C++ Source Code Examples

Example C/C++ Code: Include Statements for DSPower-HWLib

Example C/C++ Code: Initializing DSP/Analog Hardware

Example C/C++ Code: Creating a Callback Window

Example C/C++ Code: Processing Buffer Ready Messages

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Example C/C++ Code: Unregistering Callback Handles to Receive Update Messages

Example C/C++ Code: Acquiring Waveform Files

Example C/C++ Code: Playing Back Waveform Files

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Below is a partial source code example from dtape.cpp showing registration of a callback window or control handle to receive update messages:

```
// register window to receive various messages from engine
DSRegisterEngineMsgWnd(hEngine, DS_REMW_SETENGINESTATUSMSG |
DS_REMW_SETSTATISTICSMSG | DS_REMW_SETBUFNUMMSG |
DS_REMW_SETLEVELMSG, hdlg);
```

Note that only one callback window or control handle can be assigned to an engine at one time. Thus, another DSRegisterEngineMsgWnd call using a new handle would essentially redirect future update messages to the new handle.

Below are partial source code examples similar to the dtape.cpp example file included with DSPower-HWLib software showing some typical processing of update messages:

```
case WM_DSPENGINE_PROGSTATUS:
```

```
        // if engine has entered idle state due to error, finishing the instructed function,
        // etc, process STOP button to reflect the current status

        if (State != IDLE && wParam == DS_PS_PROGDONE) {

            SendMessage(hdlg, WM_COMMAND, DS_ATC_TAPESTOP, 0L);
        }
        break;

case WM_DSPENGINE_SAMPFREQ:

        // get IEEE floating-point actual sampling rate from engine message

        float Fs;
        _fmemcpy(&Fs, &lParam, sizeof(float));  // engine sends IEEE single-float value

        char tmpstr[30];
        gcvt(Fs, 10, tmpstr);  // convert f.p. value to string

        UpdateStatusBar(hSamplingRate, tmpstr, "Actual Sampling Rate");
        break;

case WM_DSPENGINE_BUFSIZE:

        dwBufferSize = lParam;  // store the buffer length being used
        break;

case WM_DSPENGINE_BUFNUM:

        dwCurSamplePos = BufferSize*lParam;  // calculate current sample count

        char tmpstr[30];
        ltoa(dwCurSamplePos, tmpstr, 10);  // convert to text

        UpdateStatusBar(hSampleCounter, tmpstr, "Current Sample Count");
        break;

case WM_DSPENGINE_LEVEL:

        UpdateMeter(hMeter, SETMETERLEVEL, lParam);  // lParam has amplitude
        break;
```

Example C/C++ Code: Unregistering Callback Handles to Receive Update Messages

Below is a partial source code example showing unregistration of window or control handles currently receiving specified types of update messages:

```
// unregister windows receiving various messages from engine
DSUnregisterEngineMsgWnd(hEngine, DS_REMW_SETENGINESTATUSMSG |
DS_REMW_SETSTATISTICSMSG | DS_REMW_SETBUFNUMMSG |
DS_REMW_SETLEVELMSG);
```

Note that a window or control handle is not needed in the DSUnregisterEngineMsgWnd call because only one callback window handle for each callback message category can be actively registered at any one time. In the example above, whatever number of window handles were registered using DSRegisterEngineMsgWnd calls are unregistered (but only for the callback message categories specified).

DSPower-HWLib and DSPower-SPLib C/C++ Source Code Examples

Example C/C++ Code: Include Statements for DSPower-HWLib

Example C/C++ Code: Initializing DSP/Analog Hardware

Example C/C++ Code: Creating a Callback Window

Example C/C++ Code: Processing Buffer Ready Messages

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Example C/C++ Code: Unregistering Callback Handles to Receive Update Messages

Example C/C++ Code: Acquiring Waveform Files

Example C/C++ Code: Playing Back Waveform Files

Example C/C++ Code: Acquiring Waveform Files

Below is a partial source code example from dtape.cpp showing typical setup and usage for the DSAcquireWvfrmFile call. The dtape.cpp demo attempts to open Hypersignal-Macro or Hypersignal-Acoustic software as a DSP Engine, but the source code example below applies regardless of the engine. One exception may occur in the szFilename parameter of the CONVERSIONINFO structure; for more information, see section 4.5 below, Acquiring Data Using DSPower-HWLib Software.

```
// create CONVERSIONINFO object memory; use zero-initialization to set most
parameters to default values

hConvInfo = LocalAlloc(LMEM_ZEROINIT | LMEM_MOVEABLE |
LMEM_DISCARDABLE, sizeof(CONVERSIONINFO));

pConvInfo = (PCONVERSIONINFO)LocalLock(hConvInfo);

lstrcpy(pConvInfo->szFilename, szWaveformFilename);  // file to acquire

// setup for one channel, init channel list (example is sequential)

pConvInfo->NumChan = 1;          // one channel
for (j=0; j<pConvInfo->NumChan; j++) pConvInfo->ChanList[j] = j;

// setup up sampling rate, amount, gain, trigger information

pConvInfo->Fs =22050;  // 22.05 kHz sampling rate
pConvInfo->NumSamples = 100000L;  // 100,000 samples
lstrcpy(pConvInfo->szGainList, "1");  // see notes in CONVERSIONINFO
pConvInfo->NumTrigChan = 1;  // one trigger channel
pConvInfo->TrigChanList[0] = 0;  // freerun trigger

// start acquisition

DSAcquireWvfrmFile(hBoard, pConvInfo);

// free and delete memory

LocalUnlock(hConvInfo);
LocalFree(hConvInfo);
```

DSPower-HWLib and DSPower-SPLib C/C++ Source Code Examples

Example C/C++ Code: Include Statements for DSPower-HWLib

Example C/C++ Code: Initializing DSP/Analog Hardware

Example C/C++ Code: Creating a Callback Window

Example C/C++ Code: Processing Buffer Ready Messages

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Example C/C++ Code: Unregistering Callback Handles to Receive Update Messages

Example C/C++ Code: Acquiring Waveform Files

Example C/C++ Code: Playing Back Waveform Files

Example C/C++ Code: Playing Back Waveform Files

Below is a partial source code example from dtape.cpp showing typical setup and usage for the DSGenerateWvfrmFile call. The dtape.cpp demo attempts to open Hypersignal-Macro or Hypersignal-Acoustic software as a DSP Engine, but the source code example below applies regardless of the engine. One exception may occur in the szFilename parameter of the CONVERSIONINFO structure; for more information, see section 4.5 below, Acquiring Data Using DSPower-HWLib Software.

```
// create CONVERSIONINFO object memory; use zero-initialization to set most parameters to default values

hConvInfo = LocalAlloc(LMEM_ZEROINIT | LMEM_MOVEABLE | LMEM_DISCARDABLE, sizeof(CONVERSIONINFO));

pConvInfo = (PCONVERSIONINFO)LocalLock(hConvInfo);

lstrcpy(pConvInfo->szFilename, szWaveformFilename);  // file to play

// NOTE:  FOR D/A OUTPUT ONLY, a 0 value for number of channels, sampling frequency, and number of samples causes engine to "auto configure", initializing these values from waveform file header

pConvInfo->NumChan = 0;
pConvInfo->Fs = 0;
pConvInfo->NumSamples = 0;
```

```
lstrcpy(pConvInfo->szGainList, "1");  // see notes in CONVERSIONINFO

// start playback

DSGenerateWvfrmFile(hBoard, pConvInfo);

// free and delete memory

LocalUnlock(hConvInfo);
LocalFree(hConvInfo);
```

DSPower-HWLib and DSPower-SPLib C/C++ Source Code Examples

Example C/C++ Code: Include Statements for DSPower-HWLib

Example C/C++ Code: Initializing DSP/Analog Hardware

Example C/C++ Code: Creating a Callback Window

Example C/C++ Code: Processing Buffer Ready Messages

Example C/C++ Code: Registering Callback Handles to Receive Update Messages

Example C/C++ Code: Unregistering Callback Handles to Receive Update Messages

Example C/C++ Code: Acquiring Waveform Files

Example C/C++ Code: Playing Back Waveform Files

4.3 DSPower-HWLib Function Overview

The subsections listed below in this section provide a brief overview of the main groups of DSPower-HWLib functions. For detailed information about each function, Signalogic DSP Software Reference Guide section 2.2, DLL Functions, is the place to look. Also, Function Alphabetic List and Cross-Reference, in the overview section of chapter 2, Function Reference, contains an alphabetized list of all functions. If you are using Signalogic DSP Software online Help, then perform the following steps:

1. select Program Manager | Reference Guide (in the Signalogic DSP Software group)
2. select "Search", and enter the first few characters of the function you are looking for
3. if you are in another online Help file besides Reference Guide, find and click on a topic starting with "Reference..." to bring up a quick reference section to Reference Guide topics

Another place to look for quick reference information on DSPower-HWLib functions are the following include files:

- enmgr.h (Engine Manager DLL)
- hwmgr.h (Hardware Manager DLL)
- hwlib.h (DSPower-HWLib DLL)

Keep in mind also that before any of the functions listed below in this section are used, DSEngineOpen or DSAssignBoard calls (or both) must have been previously invoked. The engine and board handles returned by these calls are needed by other functions in DSPower-HWLib. For additional information, section 4.1.1, Engine and Board Handles in DSPower-HWLib and DSPower-SPLib Software.

Main DSPower-HWLib Function Groups

4.3.1 Hardware and Processor Control

Board and processor control functions are the lowest-level function available in the DSPower-HWLib DLL. Some of these functions essentially amount to "touching" DSP/analog hardware directly. Below is a partial list of these functions.

Board and Processor Control Functions

Initialize: DSInitBoard

Reset: DSResetBoard and DSResetProcessor

Run: DSRunBoard and DSRunProcessor

Run: DSRunBoard and DSRunProcessor

Hold State: DSHoldBoard and DSHoldProcessor

Shutdown: DSDisableBoard

Related Topics

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

DSP Memory Data Transfers

Executing High-Level Functions

Initialize: DSInitBoard

The DSInitBoard call first checks for the presence of the specified DSP/Analog hardware. If found, it resets all accessible registers, and places the processor into a known state. Usually this simply means the DSP processor is held in reset; however, for some hardware, a short "host interface" segment of code is downloaded and the processor is left running. In these instances, the short code segment usually involves onchip DMA transfer from host PC "communication registers", on an interrupt-driven basis. These cases occur for DSP/analog hardware types which do not support direct access to onboard DSP memory; example hardware types include DSP5600x, and a few TMS320C3x, 'C4x, and 'C5x.

For multiprocessor hardware, all processors are initialized identically (and simultaneously, if the hardware supports this mode).

Board and Processor Control Functions

Initialize: DSInitBoard

Reset: DSResetBoard and DSResetProcessor

Run: DSRunBoard and DSRunProcessor

Run: DSRunBoard and DSRunProcessor

Hold State: DSHoldBoard and DSHoldProcessor

Shutdown: DSDisableBoard

Related Topics

Example C/C++ Code: Initializing DSP/Analog Hardware

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

DSP Memory Data Transfers

Executing High-Level Functions

Reset: DSResetBoard and DSResetProcessor

The DSResetBoard call places the DSP device in reset. Other registers and resources are not affected.

For multiprocessor hardware, DSResetProcessor resets all processors (simultaneously, if the hardware supports this mode).

The DSResetProcessor version is intended for multiprocessor hardware and has an additional parameter which allows any combination of processors on the DSP/analog hardware to be placed in reset.

Board and Processor Control Functions

Initialize: DSInitBoard

Reset: DSResetBoard and DSResetProcessor

Run: DSRunBoard and DSRunProcessor

Run: DSRunBoard and DSRunProcessor

Hold State: DSHoldBoard and DSHoldProcessor

Shutdown: DSDisableBoard

Related Topics

Example C/C++ Code: Initializing DSP/Analog Hardware
Hardware and Processor Control
DSP Program File Downloading
Hardware Interrogation
DSP Memory Data Transfers
Executing High-Level Functions

Run: DSRunBoard and DSRunProcessor

The DSRunBoard call removes the DSP device from a reset state, allowing it to run. Typically, this means the device will execute its start-up reset sequence, including jumping the code location specified in its reset vector.

For multiprocessor hardware, DSRunBoard allows all processors to run (simultaneously, if the hardware supports this mode).

The DSRunProcessor version is intended for multiprocessor hardware and has an additional parameter which allows any combination of processors on the DSP/analog hardware to be allowed to run.

Board and Processor Control Functions

Example C/C++ Code: Initializing DSP/Analog Hardware
Initialize: DSInitBoard
Reset: DSResetBoard and DSResetProcessor
Run: DSRunBoard and DSRunProcessor
Run: DSRunBoard and DSRunProcessor
Hold State: DSHoldBoard and DSHoldProcessor
Shutdown: DSDisableBoard

Related Topics

Hardware and Processor Control
DSP Program File Downloading

Hardware Interrogation

DSP Memory Data Transfers

Executing High-Level Functions

Hold State: DSHoldBoard and DSHoldProcessor

The DSHoldBoard call places the DSP device in hold state, if it supports this mode. Currently, hold state is applicable only for some DSP/analog hardware with TMS320Cxx devices, and even some of these cases, hold state is not supported by the board design.

For multiprocessor hardware, DSHoldBoard places all processors in hold state (simultaneously, if the hardware supports this mode).

The DSHoldProcessor version is intended for multiprocessor hardware and has an additional parameter which allows any combination of processors on the DSP/analog hardware to be placed in hold state.

Board and Processor Control Functions

Initialize: DSInitBoard

Reset: DSResetBoard and DSResetProcessor

Run: DSRunBoard and DSRunProcessor

Run: DSRunBoard and DSRunProcessor

Hold State: DSHoldBoard and DSHoldProcessor

Shutdown: DSDisableBoard

Related Topics

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

DSP Memory Data Transfers

Executing High-Level Functions

Shutdown: DSDisableBoard

The DSDisableBoard call places the DSP device in reset, and disables any additional resources on the board that may be used by the DSP code driver. Examples could include disabling resources accessible from the host PC, such sampling rate generation timers, or issuing a "shutdown" interrupt to the DSP code which causes it to disable resources not accessible from the PC.

For multiprocessor hardware, DSDisableBoard resets all processors (simultaneously, if the hardware supports this mode).

Board and Processor Control Functions

Initialize: DSInitBoard

Reset: DSResetBoard and DSResetProcessor

Run: DSRunBoard and DSRunProcessor

Run: DSRunBoard and DSRunProcessor

Hold State: DSHoldBoard and DSHoldProcessor

Shutdown: DSDisableBoard

Related Topics

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

DSP Memory Data Transfers

Executing High-Level Functions

4.3.2 DSP Program File Downloading

The DSLoadFileBoard call downloads a specified executable DSP file to the all processors on the DSP/analog hardware.

The DSLoadFileProcessor version is intended for multiprocessor hardware and has an additional parameter which allows any

combination of processors to be specified to receive the executable DSP file. Note that in some cases, even if not specified in the call, it may be unavoidable downloading executable files to DSP devices on a multiprocessor board because they may share program memory. Such a case does not necessarily mean that processors are running the same program, only that programs being run by another processor may be "visible" somewhere in program memory.

Related Topics

Example C/C++ Code: Initializing DSP/Analog Hardware

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

DSP Memory Data Transfers

Executing High-Level Functions

4.3.3 Hardware Interrogation

Board interrogation functions allow DSP/analog hardware drivers to report back important information about the hardware. This can involve some sophisticated detection on the part of the drivers, such as determining how much memory has been installed on the DSP board. Below is a partial list of these functions.

Hardware Interrogation Functions

Memory Size: DSGetMemSize

Memory Architecture: DSGetMemArch

Word Length: DSGetWordLength

Sampling Frequency Calculation: DSCalcSampFreq

Hardware Classification: DSGetBoardClass

Related Topics

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

DSP Memory Data Transfers

Executing High-Level Functions

Memory Size: DSGetMemSize

The DSGetMemSize function returns the amount of data memory detected on the specified DSP processor on the specified DSP/analog hardware. For most hardware types, the return value is in units of the "natural wordlength" of the DSP device. Some examples include 32-bit words for TMS320C3x and TMS320C4x devices, 24-bit words for DSP5600x devices, and 16-bit words for ADSP-21xx and TMS320C5x devices. However, there are currently two (2) exceptions. The DSGetMemSize return value is in 16-bit words for DSP32C and DSP3210 devices, even though these devices are 32-bit floating-point. This is because these devices have true multiwidth access to onchip and external memory, and some DSP Engine drivers take advantage of this feature to store data (especially analog I/O data) in compact form when possible.

Notes

1. In cases where the DSP/analog hardware has separate program and data memory areas, the DSGetMemSize function does not measure program memory.

2. In some cases, the return value may either be somewhat less than or exceed the amount of installed onboard, external data memory. This may be because external memory addressing is contiguous with onchip memory, or overlaps onchip memory (creating an unusable region).

Hardware Interrogation Functions

Memory Size: DSGetMemSize

Memory Architecture: DSGetMemArch

Word Length: DSGetWordLength

Sampling Frequency Calculation: DSCalcSampFreq

Hardware Classification: DSGetBoardClass

Related Topics

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

DSP Memory Data Transfers

Executing High-Level Functions

Memory Architecture: DSGetMemArch

The DSGetMemArch function returns the type of memory architecture used by the DSP device on the DSP/analog hardware specified by the input board handle. Following are the possible return values:

DS_GMA_LINEAR	linear, both program and data in the same memory space
DS_GMA_HARVARD	modified Harvard architecture, with program and data linear in separate memory spaces
DS_GMA_VECTOR	modified Harvard architecture with program and data in separate memory spaces, and with data memory further subdivided into vector memory spaces

Notes

1. This function does not actually "touch" hardware. The return value is completely determined by what the information in the Hardware Setup File entry for the specified DSP/analog hardware type.

Hardware Interrogation Functions

Memory Size: DSGetMemSize

Memory Architecture: DSGetMemArch

Word Length: DSGetWordLength

Sampling Frequency Calculation: DSCalcSampFreq

Hardware Classification: DSGetBoardClass

Related Topics

Example C/C++ Code: Processing Buffer Ready Messages

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

DSP Memory Data Transfers

Executing High-Level Functions

Word Length: DSGetWordLength

[@j natural wordlength]

The DSGetWordLength function returns the natural word length used for data memory by the DSP device on the DSP/analog hardware specified by the input board handle. Following are the possible return values:

DS_GMA_SIZE16	indicates 16-bit fixed-point data memory
DS_GMA_SIZE24	indicates 24-bit fixed-point data memory
DS_GMA_SIZE32	indicates 32-bit floating-point data memory

Notes

1. This function does not actually "touch" hardware. The return value is completely determined by what the information in the Hardware Setup File entry for the specified DSP/analog hardware type.
2. This function is convenient for use with DSGetMem or DSPutMem functions when transferring integer data between DSP data memory and host PC memory.

Hardware Interrogation Functions

Memory Size: DSGetMemSize

Memory Architecture: DSGetMemArch

Word Length: DSGetWordLength

Sampling Frequency Calculation: DSCalcSampFreq

Hardware Classification: DSGetBoardClass

Related Topics

Example C/C++ Code: Processing Buffer Ready Messages

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

DSP Memory Data Transfers

Executing High-Level Functions

Sampling Frequency Calculation: DSCalcSampFreq

Given a desired sampling frequency, the DSCalcSampFreq function return two important pieces of information:

1. A mode or control register value that can be sent to resident DSP code and used to program the DSP/analog hardware sampling rate clock generators. High-level functions in DSPower-HWLib use this value implicitly.
2. The actual sampling rate that the DSP/analog hardware will use, given the desired sampling frequency as input. This value is important because often times only a limited range of discrete sampling rates are possible.

Below is an example of using the DSCalcSampFreq call:

```
DWORD FsMode;
float ActualFs;
FsMode = DSCalcSampFreq(hBoard, 44100, NULL, NULL, &ActualFs);
```

The example above asks for a sampling rate of 44.1 kHz, and assumes simultaneous sampling, which requires no entry for number of

channels and channel list (see notes below). The closest sampling frequency value to 44.1 kHz that can be achieved by the hardware is returned in ActualFs, and the corresponding value needed to program the hardware's clock generator is returned in FsMode.

Notes

1. The actual sampling rate is returned in a float value pointed to by a function call parameter, while the mode value is the function return value.

2. The number of channels and channel list input parameters are significant only for DSP/analog hardware which does not utilize simultaneous sampling for all channels. Examples include "channel scan" and multiplexing channel groups into single A/D converters. If you know the specified DSP/analog hardware has simultaneous sampling in all cases, the number of channels parameter and channel list parameter can be left NULL.

Hardware Interrogation Functions

Memory Size: DSGetMemSize

Memory Architecture: DSGetMemArch

Word Length: DSGetWordLength

Sampling Frequency Calculation: DSCalcSampFreq

Hardware Classification: DSGetBoardClass

Related Topics

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

DSP Memory Data Transfers

Executing High-Level Functions

Hardware Classification: DSGetBoardClass

The DSGetBoardClass function returns a hardware classification value suitable for use by DSP Source Code Interface software. This value is maintained by DSP Engines because DSP Source Code Interface software for one type of DSP chip family may support multiple types of DSP/analog hardware, each with different configuration and analog I/O driver characteristics. It can be sent to executing DSP source code during or prior to run-time initialization. See Signalogic DSP Software Reference Guide section 3.2, Board Designator, Driver ID, and Classification Table.

The result of the DSGetBoardClass function can be sent directly to the Hypersignal DSP Source Code software DSP_BOARDCLASS and DSP_BOARDSUBCLASS variables. See Signalogic DSP Software Reference Guide section 3.2, Board Designator, Driver ID, and Classification Table.

Hardware Interrogation Functions

Memory Size: DSGetMemSize

Memory Architecture: DSGetMemArch

Word Length: DSGetWordLength

Sampling Frequency Calculation: DSCalcSampFreq

Hardware Classification: DSGetBoardClass

Related Topics

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

DSP Memory Data Transfers

Executing High-Level Functions

4.3.4 DSP Memory Data Transfers

Data transfer functions allow C/C++ and MATLAB programs to transfer data between local or global host PC memory and DSP onchip or external (onboard) memory. The workhorse data transfer functions in DSPower-HWLib are the DSPutMem and DSGetMem calls. These functions allow one or more data elements of variable length to be transferred to/from various segments of DSP memory. All types of available DSP memory are accessible, with the possible exception of DSP chip onchip memory, which may only accessible from the DSP device under certain circumstances.

Also available are the DSPutHVarMem and DSGetHVarMem calls, which allow pre-defined variables and semaphores used in Hypersignal DSP Source Code Interfaces to be transferred to and from DSP memory. For more information, see section 5.0.2, DSP Source Code Interfaces for Hypersignal.

Data Transfer Functions

Memory Block Transfers: DSPutMem and DSGetMem

Hypersignal DSP Code Variable Transfers: DSPutHVarMem and DSGetHVarMem

Related Topics

Example C/C++ Code: Processing Buffer Ready Messages

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

DSP Memory Data Transfers

Executing High-Level Functions

Memory Block Transfers: DSPutMem and DSGetMem

The DSPutMem and DSGetMem calls allow data elements to be transferred between host PC memory and DSP/analog hardware memory. One or more data elements can be transferred to or from a specified address in DSP external memory, or in onchip DSP memory for some types of hardware. Depending on the hardware, it may be possible to control the width of the data elements. Below are some examples:

```
if (uMemArch == DS_GMA_VECTOR) {  // vector data memory

    uStatus = DSGetMem(hBoard, DS_GM_VECTOR_DATA_X, Addr,
    DS_GM_SIZE16, pBuf, wBuflen);
}
else { // linear data memory

    uStatus = DSGetMem(hBoard, DS_GM_LINEAR_DATA_RT, Addr,
    DS_GM_SIZE16, pBuf, wBuflen);
}
```

The above examples show reading from DSP data memory at a location specified by Addr. wBuflen data elements of 16-bits each are read. The first example assumes a hardware type with vector data memory, and the second linear data memory (see Memory Architecture: DSGetMemArch in section 4.3.3 above, Hardware Interrogation). The return value should be checked for success.

Data Transfer Functions

Memory Block Transfers: DSPutMem and DSGetMem

Hypersignal DSP Code Variable Transfers: DSPutHVarMem and DSGetHVarMem

Related Topics

Example C/C++ Code: Processing Buffer Ready Messages

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

Hypersignal DSP Code Variable Transfers: DSPutHVarMem and DSGetHVarMem

The DSPutHVarMem and DSGetHVarMem functions allow variables and semaphores to be transferred between C/C++ and MATLAB programs and pre-defined locations maintained by Hypersignal DSP Source Code Interfaces for various DSP devices. Hypersignal DSP Source Code Interfaces include several real-time operating modes, as well as a number of math and DSP routines. For detailed information, see section 5.0.2, DSP Source Code Interfaces for Hypersignal.

Below are some examples:

```
DSPutHVarMem(hBoard, DSP_BOARDCLASS, wBoardClass & 0x0ff);
DSPutHVarMem(hBoard, DSP_OPMODE, 2);  // Dig. Scope is mode 2
DSPutHVarMem(hBoard, DSP_BUFLEN, wBuflen);  // buffer size
DSPutHVarMem(hBoard, DSP_NUMCHAN, 1);     // number of channels
DSPutHVarMem(hBoard, DSP_FSMODE, FsMode);  // sampling rate mode value
```

The above examples are taken from the dscope.cpp example file; in the this file, hBoard is previously created by a DSAssignBoard call to a DSP/analog hardware type chosen from the Hardware Selector. The first example writes the board classification returned by DSGetBoardClass, to indicate to the Hypersignal DSP source code what hardware is being used. Succeeding examples set up the operating mode, buffer length, number of channels, and sampling rate clock generation value (see DSCalcSampFreq). These examples are only small portion of what is needed to completely enable one of the built-in operating modes in the Hypersignal DSP Source Code Interfaces.

Data Transfer Functions

Memory Block Transfers: DSPutMem and DSGetMem

Hypersignal DSP Code Variable Transfers: DSPutHVarMem and DSGetHVarMem

Related Topics

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

DSP Memory Data Transfers

Executing High-Level Functions

4.3.5 Executing High-Level Functions

High-level functions in DSPower-HWLib are defined as functions which are handled completely by the DSP Engine, given a minimal set of input parameters, with no monitoring or updating required by the calling application. For example, an engine might be instructed to acquire a waveform file of a specified length with specified trigger criteria.

Currently, there are two types of high-level engine functions available in DSPower-HWLib software:

- data acquisition and measurement functions
- general DSP Engine functions

Data acquisition and measurement functions are implemented by first initializing a CONVERSIONINFO structure, and then passing a pointer to this structure as a parameter to the DSAcquireWvfrmFile or DSGenerateWvfrmFile functions. These flexible functions allow simple operations such as acquiring and playing back waveform files, as well as more complicated operations such as stimulus & response measurement and real-time digital filtering. See section 4.5 below, Acquiring Data Using DSPower-HWLib Software.

General DSP Engine functions can be executed using the DSExecEngineFunction found in Engine Manager. Engine functions can be executed synchronously or asynchronously, meaning the calling application either waits for the engine function to complete, or continues executing simultaneously. The DSExecEngineFunction allows any DSP Engine function to be executed, using a command string--essentially as if the engine was being operated in user mode.

Making DSP Engine functions available in this way offers great flexibility, but not great ease-of-use. Unlike the data acquisition and measurement functions available from DSPower-HWLib, to successfully use DSExecEngineFunction, you need to know the details of the engine function you are trying to invoke. For more information, see section 7.3, Executing Arbitrary Functions in DSP Engines.

Related Topics

Example C/C++ Code: Acquiring Waveform Files

Example C/C++ Code: Playing Back Waveform Files

Hardware and Processor Control

DSP Program File Downloading

Hardware Interrogation

DSP Memory Data Transfers

Executing High-Level Functions

4.4 DSPower-SPLib Function Overview

This section not available yet.

4.5 Acquiring Data Using DSPower-HWLib Software

Analog data can be acquired (or output) to/from supported DSP/analog hardware using DSPower-HWLib software in the following ways:

1. Using the DSAcquireWvfrmFile and DSGenerateWvfrmFile calls in hwlib.dll. For examples, see Example C/C++ Code: Playing Back Waveform Files, and Example C/C++ Code: Acquiring Waveform Files, in section 4.2.4 above, Example Code: DSPower-HWLib and DSPower-SPLib Software.

2. Using the DSGetMem or DSPutMem calls in hwlib.dll, assuming that an appropriate DSP Program File has been downloaded to the DSP/analog hardware beforehand.

3. Executing a DSP Engine function that acquires or generates analog I/O data to/from waveform file or DSP board memory in some other way.

Related Topics

CONVERSIONINFO Structure

Example C/C++ Code: Acquiring Waveform Files

Example C/C++ Code: Playing Back Waveform Files

General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software

Creating Programs for DSPower-HWLib and DSPower-SPLib

DSPower-SPLib Function Overview

Acquiring Data Using DSPower-HWLib Software

Using Engine Manager

Using Hardware Manager

CONVERSIONINFO Structure

[@j CONVERSIONINFO structure, CONVERSIONINFO, PCONVERSIONINFO]

Below is a reconstruction of the CONVERSIONINFO structure, which is used in high-level DSPower-HWLib calls such as DSAcquireWvfrmFile and DSGenerateWvfrmFile. The reprint below is partially divided into the following sections:

- basic acquisition/generation information
- real-time digital filter information
- trigger information
- stimulus & response (simultaneous input/output information)

Each element in the structure is commented with a brief description. For more detailed information, see the additional explanations of some elements after the structure.

typedef struct {

// basic acquisition/generation information

// type	**// element**	**// brief description**
char	szFilename[144];	// [path]filename of file being acquired/generated (see notes below)
short int	NumChan;	// number of channels
short int	ChanList[MAXCHAN];	// channel list, maps physical channels to waveform traces (entries should not exceed NumChan)
float	Fs;	// sampling rate (in Hz)
DWORD	NumSamples;	// number of samples to acquire/generate
char	szGainList[41];	// string specifying gain list for input channels; can contain ',' and '-' other characters to specify lists and ranges (see notes below)
long	Offset;	// offset added to data (in A/D counts, can be +/-)
BOOL	DALooping;	// enables/disables continuous D/A looping; 0 or 1, valid only during generation (output)
BOOL	Monitor;	// enables/disables D/A loopback when recording; 0 or 1, valid only during acquisition (input)
BOOL	FileSplit;	// 1 enables A/D file-splitting for multichannel, 0 disables
DWORD	Frmsiz;	// framesize (stored in waveform header; used for post-processing only)

// real-time digital filter information

// type	**// element**	**// brief description**
char	szFilter1[13];	// filename of real-time digital filter file; can contain '/' character
char	szFilter2[13];	// to allow dual-filter entry (filter files must be located on same path as waveform)

// trigger information

// type	**// element**	**// brief description**
WORD	TrigMode;	// trigger mode (0 = internal, 1 = external)
long	TrigLevel;	// trigger level (in A/D counts, can be +/-)
long	TrigDelay;	// delay from trigger relative to data (in sec, can be ±)
short int	NumTrigChan;	// number of trigger channels

```
short int       TrigChanList[MAXCHAN];   // trigger channel list (entries should not
                                            exceed NumTrigChan
// stimulus & response information
// type         // element                // brief description
char            szStimName[13];          // name of stimulus output file; used only
                                            for DSStimRespMeasure function (must be
                                            located on same path as waveform)
WORD            StimMode;                // controls stimulus usage in
                                            Stimulus/Response function
long            StimDelay;               // delay from stimulus onset to start of
                                            record (+/- value)
} CONVERSIONINFO;
typedef CONVERSIONINFO FAR* PCONVERSIONINFO;
```

Notes

1) The entire CONVERSION structure should be initialized to zero before using, so that default values can take effect for parameters not used. This can easily be accomplished using a call such as

```
_fmemset(&ConversionInfo, 0, sizeof(CONVERSIONINFO));
```

2) Previous value used in either DSEngineOpen or DSSetEngineWaveformPath call is used for waveform filenames, unless explicit path information is found in the filename above.

NOTE: For operation when a Hypersignal-Macro series program is the DSP Engine, szFilename above should not contain path information. The path should instead be set using the initial DSEngineOpen call, or a subsequent DSSetEngineWaveformPath call.

3) The szGainList parameter is a string because input (or output) gain and/or attenuation can be specified as a software digital scaling value, or as a hardware gain value in Volts or dB, depending on the DSP/analog hardware type. Also, entry can be channel-specific, or specified as a channel-range.

4) TrigDelay parameter not yet operational.

5) PCONVERSIONINFO can be used to declare pointers to CONVERSIONINFO structures. For examples, see Example C/C++ Code: Playing Back Waveform Files, and Example C/C++ Code: Acquiring Waveform Files, in section 4.2.4 above, Example Code: DSPower-HWLib and DSPower-SPLib Software.

Related Topics

CONVERSIONINFO Structure

Example C/C++ Code: Acquiring Waveform Files

Example C/C++ Code: Playing Back Waveform Files

General Topics

Architecture of DSPower-HWLib and DSPower-SPLib Software

Creating Programs for DSPower-HWLib and DSPower-SPLib

DSPower-SPLib Function Overview

Acquiring Data Using DSPower-HWLib Software

Using Engine Manager

Using Hardware Manager

Notes: